Defining product architectures.

Eric Bonjour, Ghassen Harmel, Maryvonne Dulmet

Defining product architectures

Eric Bonjour, Ghassen Harmel and Maryvonne Dulmet

Département Automatique et Systèmes Micro-Mécatroniques (AS2M)
Institut FEMTO-ST, UMR CNRS 6174 - UFC / ENSMM / UTBM
24, rue Alain Savary 25000 Besançon (France)- ebonjour@ens2m.fr

Abstract

Identifying product architectures is recognized as a critical activity during the preliminary design phase since the selected architecture has a great impact on the product quality, on the organization of the following phases of the design project and on the global performance of this project. System architects need methods to pre-determine cohesive modules and integrative elements. In this paper we present a new method to jointly design the functional and physical architectures. Starting from an incidence matrix that represents the couplings between elements of two domains (here, functions and components), it generates a Design Structure Matrix (DSM) for each domain. Each DSM is then rearranged by a clustering algorithm in order to reveal a "satisfactory" architecture. To illustrate our approach, we present an industrial case study in the framework of a new automobile engine development project.

Keywords: modular design, product architecture, DSM

1. Introduction

Modular product design has proved to be an efficient strategy to obtain scale economy and to reduce design efforts. According to Ulrich [1], the product architecture is the mapping of the product functions onto its components. Optimal modular product architectures can be defined as the clustering of components such that the degree of interaction/dependency is maximised within groups (or modules) and minimised between groups (inter-modules) [2]. Modules are commonly described as groups of functionally or structurally dependent components. Research, concerning platform-based product development and product family design [3, 4], has received huge interest over the last decade since it aims at providing methods to identify common modules and generate product variants with distinctive modules (commonality vs variety). Modularity has many advantages but few methods exist to partition a product into modules, even in the special case of single complex products. Ulrich [1] defines product architectures as "the scheme by which the function of a product is allocated to physical components." A key feature of product architecture is the degree to which it is modular or integrative [5]. Sharman and Yassine [6] point out that modularity has drawbacks. The inter-module interfaces must allow change to occur within modules without adversely affecting inter-module working. This requires an appropriate definition of interfaces that play the key role of connecting and interacting between components.

The system architect and the teams (s)he manages need formal representations in order to address interactions between elements in the system [7]. When couplings between the elements of product domains have not been formally addressed, the integration of the teams' contributions is more difficult and requires numerous design iterations. Few architecting methods have been developed to identify modular product architecture. They use different product architecture representations [6] for instance, diagrams [8], oriented graphs [9] or matrices. The inputs of these methods may be either functional models [8], or components interactions [10], [11], [12], or a mapping of functions onto physical

components [13], or more complex data intended to take into account key factors of the whole life cycle of the system [14]. Sosa et al. (2003) [11] introduce the concepts of modular and integrative systems, from an external perspective, that is, based on the existence of design interfaces between components of the same product. They define "modular systems as those whose design interfaces with other systems are clustered among a few physically adjacent systems, whereas integrative systems are those whose design interfaces span all or most of the systems that comprise the product due to their physically distributed or functionally integrative nature throughout the product." Following this definition, we use the terms "module" and "integrative elements" in the remainder of this paper.
In this paper, we propose to provide the system architects with a tool that supports the generation of both functional and physical product architectures. First, we review matrix-based methods for product architecture modelling. Second, we propose a fuzzy method to simultaneously generate the functional and physical architectures of the product. Finally, we present and interpret the results obtained with this method concerning the preliminary layout of an automotive engine block, along with further works.

2. DSM and DMM as product architecture modelling tools

Matrix-based product modelling methods are being increasingly used [6], since they could support different research goals: for example, product modularisation [10], analysis of technical interactions either within the products or within the project organization [11], and change propagation analysis [15]. System architecture modelling relies on two kinds of matrix in order to design a complex system in a systematic and coherent way: Domain Mapping Matrix, DMM, and Design Structure Matrix, DSM.

A DMM represents relationships between two domains. These matrices are basically incidence matrices. They can represent a set of design decisions or relationships between what and how. One example is the Axiomatic Design Matrix [16], which captures the relationships between a function, and the design parameter that realises the function. Another example is the Quality Function Deployment (QFD) method [17] that uses several inter-domains matrices to convey the customer requirements throughout the development project. In [18], the authors use binary function-component matrices in order to obtain morphological matrices by using matrices multiplication. In [19], we find a very recent use of Domain Mapping Matrices (DMM), we can notice that the authors apply clustering directly on DMM.
We assume in our research work that the incidence matrices are of high importance since they have to ensure the cohesion between the product sub-domains and more generally, between project domains (Product, process and Organisation) [20].

DSM [21] are now popular modelling and analysis tools, especially for purposes of decomposition and integration. DSM display the relationships between elements of a system in a compact and visual format. DSM can be applied on various levels of abstraction to study interactions between requirements, between functions, between sub-systems or components, and between design parameters. They are used to analyse project domain architectures: the architecture of products, the architecture of design process or the decomposition of the projects into different teams [11]. A DSM is a square matrix with identical elements in rows and columns. Cells along the diagonal have no sense. Reading across a row reveals what other elements the element in that row provides to. Scanning down a column reveals what other elements the element in that column depends on.

3. Case study: an engine development project

In this part, we present an industrial case that concerns a diesel engine architecture development in a French car manufacturer and that will illustrate the interests of the method we propose. In this paper, we focus on the product domains architectures but this method could be applied on other project domains.
Either, in the conceptual design that establishes functional structures or in the preliminary embodiment design that develops the preliminary layouts [22] a crucial design activity corresponds to the mapping

between two domains and then, the choice of the preliminary architectures. The main architecture decisions are represented by the inter-domains matrices: requirements - Systems Functions incidence matrix; System Functions - Product Components (SF-PC) incidence matrix. The cells of this latter matrix correspond to coupling weights between functions and components: a function may be allocated to components that contribute to this function.
Generally speaking, incidence matrices either exist or need to be formalised. We will generate them by asking the system architect to fill in the incidence matrices with numerical values. The values are ranged from 0, for no coupling between the SF in column and the PC in row, to 9 for a total coupling between the parameters and attributes defining the SF and the PC.

	SF	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PC	System Functions / Components incidence matrix	Air Providing	Fuel Providing	Gas Cleaning-up	Combustion	Pressure-Torque Conversion	Friction	Lubrication	Power Transmission	Secondary Energy Conversion	Ventilation	Vibration	Control	Thermics	Coupling	Functional Volumes
1	EGR	9	0	9	0	0	0	0	0	0	0	0	9	9	0	7
2	Fuel System	0	8	0	8	0	7	5	9	8	0	9	8	6	0	8
3	Breech Block	8	9	6	7	0	0	8	9	5	9	5	0	9	0	8
4	Air Intake	7	0	5	9	0	0	0	0	0	6	8	7	9	0	8
5	Exhaust	8	0	8	0	0	0	4	8	0	0	8	7	8	0	7
6	Camshaft/Valve Train	9	0	0	0	0	8	8	4	9	0	5	0	6	0	8
7	Crankshaft	0	0	0	9	7	9	7	6	6	0	8	5	8	8	9
8	Casing	0	0	0	6	9	8	9	8	7	9	7	0	8	9	9
9	Lubrication and Blow-by	0	0	0	0	8	7	9	0	0	7	0	4	8	0	7
10	Accessory Drive	0	0	0	0	7	6	0	9	7	0	7	0	4	0	8
11	Synchronous Drive	0	0	0	0	8	5	0	7	8	0	7	0	6	0	7
12	Vacuum Circuit	8	0	0	0	0	0	4	0	9	6	0	0	0	0	8
13	Cooling Circuit	0	0	5	0	0	0	4	0	5	0	6	0	7	0	7
14	Secondary Energy Generator	0	0	0	0	0	4	0	5	9	0	5	0	5	0	7
15	Sensors and Control	7	9	7	0	0	0	6	0	0	0	0	7	4	0	4

Table 1. Numerical SF-PC incidence matrix

Table 1 shows a SF-PC incidence matrix concerning an engine block. It is a numerical matrix with 15 elements in rows corresponding to 15 components that make up the engine block and 15 elements in columns corresponding to 15 SF (obviously, it is a coincidence if the number of PC equals the number of SF). More precisely, we can read by scanning down the "Fuel providing" column that this function constrains several components: Fuel System, Breech Block and Sensors. In the same way, we can read through the "Fuel System" row that this component is constrained by (or contributes to) several SF: Fuel Providing, Combustion, Friction, Lubrication, Power Transmission, Vibration, Control, Thermics and Functional Volumes.

In this paper, we propose a method to simultaneously design two product domains architectures. We focus on the SF-PC incidence matrix that supports the design of the functional and physical architectures.

4. From one incidence matrix (or DMM) to 2 DSM: a fuzzy method

The generation of a SF DSM and the generation of a PC DSM are symmetrical. Thus we will explain only how to generate the SF DSM.

4.1. Axioms

In this section, we explain the axioms which are at the basis of our method, along with the underlying assumptions.

- Axiom 1. If two System Functions SF1 and SF2 constrain a component PCi then SF1 and SF2 are coupled through that component (the symmetric axiom is: "If two Product Components PC1 and PC2 contribute to the fulfilment of SFj then PC1 and PC2 are coupled through that function").
- Axiom 2. The intensity of the coupling between SF1 and SF2 (respectively PC1 and PC2) is related to the intensity of their coupling with the component PCi (respectively SFj).

Let us consider a component defined through a limited set of parameters. We distinguish between two situations. First, both SF1 and SF2 constrain a subset of parameters of the component PCi. In that case, the coupling between the two SF is clear and negotiations need to be carried out. Conversely, if there is no overlapping constrains between SF1 and SF2 for the design of all components, we assume that the two SF are mutually independent (no coupling) regarding the component domain. Second, if we consider that a given System Function SFi has a weak impact on the component PCj (that means that the value of coupling between SFi and PCj is low in the SF-PC incidence matrix), so there is a low probability that SFi impacts the same parameters describing PCj than the other SFs, that is, we assume that the coupling between SFi and the other SFs is low through the component PCj.

4.2. A fuzzy method

The fuzzy inference system we proposed is based on five stages that we will develop in this section.
We will generate one SF DSM for each component by identifying the System Functions impacting each component and by applying the rules introduced by the axioms and translated into a fuzzy inference system. We will then aggregate the resulting DSM and filter the aggregate DSM to delete meaningless values. A clustering algorithm is used to identify modules and integrative elements. The results are displayed to allow the architect to interpret the proposed architectures.

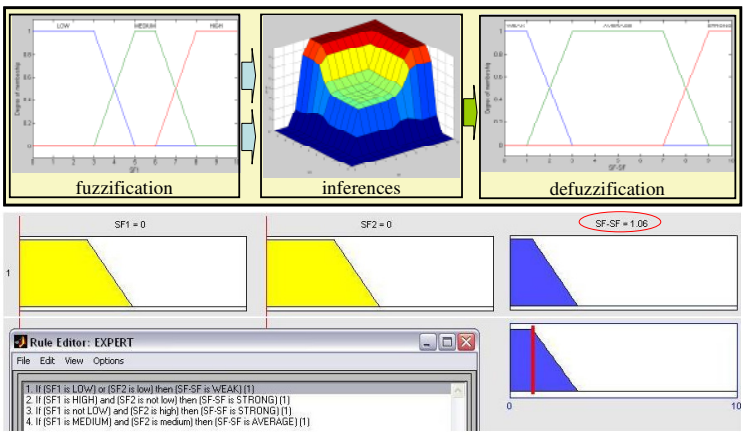

Figure 1. A fuzzy process

Step 1: build a numerical SF-PC incidence matrix

The incidence matrix has to be filled in with the system architect and his (her) team. Discussions may be necessary to identify if the interactions really exist and to estimate each incidence value. Since intensity values may be imprecise and subjective, we are clearly in a context where the use of Fuzzy Logic is relevant [23].

Step 2: for each PC, generate one SF DSM by a fuzzy process

The axioms presented in Section 4.1 are at the basis of the generation of one SF DSM per component using a Mamdani fuzzy inference system [23] for analyzing the inputs through a set of 'if-then' rules and a

T-norm aggregation operator. So if the number of components in the incidence matrix is Npc then this step generates Npc "System Function DSM". Then we use an aggregate method to obtain one DSM only (the average of all contributions). The fuzzy processing is made up of three basic stages (Figure 1).

The "fuzzification" stage (see Figure 1, left side) corresponds to the transformation of a numerical value through fuzzy variables (input). We choose the structure of the membership functions characterising the two inputs, by taking into account the architect's reasoning. We define three linguistic variables which are Low, Medium and High and we use the most common membership function, a trapezoidal function.

The fuzzy 'if-then' rules are developed to relate input to output variables. These rules represent the expert's knowledge about the interactions between input variables and their effects on the output. The inference system approximates the way an architect estimates the coupling between two SF (SF_j and SF_k) through one component PC_i. It is based on the following set of rules (see at the centre of Figure 1 - issued from Matlab Toolbox):

1. IF (SF_j-PC_i is Low) OR (SF_k-PC_i is Low) THEN (SF_j - SF_k is Weak)
2. IF (SF_j-PC_i is High) AND (SF_k-PC_i is NOT Low) THEN (SF_j - SF_k is Strong)
3. IF (SF_j-PC_i is NOT Low) AND (SF_k-PC_i is High) THEN (SF_j - SF_k is Strong)
4. IF (SF_j-PC_i is Medium) AND (SF_k-PC_i is Medium) THEN (SF_j - SF_k is Average)

Our goal has been to generate an understandable inference system for the architect. So we intentionally limit the number of rules and hence, the number of linguistic variables.

A trapezoidal membership function is used for the fuzzy logic output linguistic variables which are Weak, Average and Strong. The "Defuzzification" stage involves finding a crisp value for the coupling using the output membership function. The aggregated fuzzy output is defuzzified using the "centroid of area" technique. The formula is given in [23]. This is the most widely adopted defuzzification method, which is reminiscent of the calculation of expected values of probability distributions.

The choice of these linguistic variables aims at limiting the influence of low inputs. Tuning the system has been done to choose the more appropriate input-output membership functions and defuzzification method. This fuzzy model has been judged satisfactory both from the architect' point of view and with the simulated results. The relevance of this fuzzy inference system will be presented later when we will interpret the results of the industrial example.

Step 3: aggregate the Npc DSM and filter

In order to obtain the aggregated DSM, we used the average method. Let $DSM(i,j)_k$ denote the coupling value between SF_i and SF_j in the DSM generated by the component PC_k. Let DSM_A denote the Aggregated DSM. Then this DSM is computed as follows (Equation 1):

$$DSM_A(i,j) = \sum_{k=1}^{Npc} \frac{DSM(i,j)_k}{Npc} \qquad (1)$$

Then we need to filter the low values in the aggregated SF DSM. The filter aims at reducing meaningless coupling values by converting them to zero:

$$if\ \ DSM(i,j) \le X\ \ then\ \ DSM(i,j) = 0 \qquad (2)$$

Where X is a parameter that has to be higher than 1.06 (minimum crisp value obtained for rule 1).

Step 4: use a clustering algorithm

The goal of a clustering algorithm is to generate modules in a systematic way by optimising an objective function. The original goal of clustering was to find similarity between elements and group them together based on a threshold of similarity between elements [24]. Recent algorithms have been developed for optimizing modular design of complex products including simulated annealing [25] and Genetic Algorithms [12]. The clustering algorithm we use in our research work is based on an algorithm developed

by Idicula [26] and improved by Thebeau [25]. Idicula's algorithm assumes an underlying directed graph model for the development effort, and uses a depth-first-search technique to solve the problem.

Step 5: display the results

Different ways for displaying the results can be used. First, the clustered DSM can be represented as a graphical matrix with diamonds like in Figures 3 and 4. We choose this representation in order to display the results related to the industrial case we develop in this paper. Direct inspection of the DSM itself is one valid way of interpreting the product architecture [6].

5. Application: Identifying an engine architecture

The fuzzy method uses a numerical inter-domains incidence matrix as input and gives two DSM as results. In our industrial example, we generate one SF DSM and one PC DSM. We observe (Figures 2 and 3) that the clustering of the 2 DSM leads to approximately the same architecture. In both cases, we obtain 2 modules and 5 integrative elements. These results partially match the architect's expectations and objectively highlight some modules already built in past projects.

The architecture obtained after clustering the SF DSM (Figure 2) is as follows:

- SF Module 1: Air providing, Fuel providing, Gas Cleaning-up and Combustion SF.
- SF Module 2: Pressure-Torque Conversion, Friction, Lubrication, Secondary Energy Conversion and Coupling SF.
- Integrative elements: Functional Volumes, Thermics, Vibration, Ventilation and Power Transmission SF.

The first module corresponds to pre-combustion system functions and groups all the functions leading to the realisation of the combustion. The second module corresponds to post-combustion system functions and groups all the functions transforming the energy of combustion into mechanical energy. The integrative SF act as functions that map together all the other SF. They create the cohesion of the entire product SF.

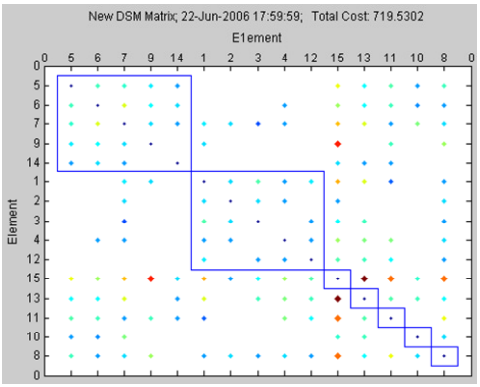

Figure 2. SF DSM with clustering

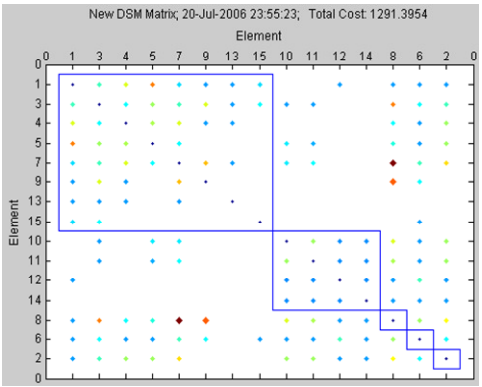

Figure 3. PC DSM with clustering

With the same parameters values for the clustering algorithm, we generate the clustered PC DSM, simultaneously with the clustered SF DSM. When presented to the architect, the clustered PC DSM (Figure 3) received a mixed reaction. But, when analysed in depth, the architect judged that the simulated architectures were coherent with the physical flows throughout the product:

- PC Module 1: The first module is composed of 6 components that directly contribute to the main external engine function, that is, to transform a thermal energy into a kinetic energy.
- PC Module 2: The second module puts together components that contribute to secondary engine functions like vacuum and electricity generating.
- There are 3 integrative elements: casing, fuel system and camshaft.

The clusters with main and secondary external functions formalise the current design process structure which gives priority to designing components realising the principal engine functions. The three integrative components had not been identified by the development team as integrative elements before we presented these results. In fact, only the casing has a physical integrative role since it links together the other engine components. But the architect judges that it is interesting to identify the fuel system and the camshaft as integrative elements since they play a central role in the engine working. These components share many constraints with the other components in reality and often these constraints are propagated by System Functions.

In idealistic modular architecture, we will find direct mapping between SF modules and PC modules. This kind of architecture ensures well decoupled modules (i.e., design sub-problems) that can be easily specified, then dealt with by different teams concurrently and reused in future system design. Unfortunately in real design situations, the modules are not completely uncoupled and in addition, there are integrative elements that link all the elements together. The consequence is that it is difficult to group a set of functions and to attach this set to a physical module, for instance for outsourcing. Our method for identifying engine architectures helps architects to question their decisions and to iterate in order to determine satisfactory SF-PC architectures.

6. Conclusions and perspectives

The tool presented in this paper transforms a given numerical SF-PC DMM into two DSM and then generates both functional and physical architectures of the product. It is based on a new fuzzy inference system and a clustering algorithm that groups elements into modules and integrative elements. The architectures generated by this tool are recommendations only but the architects should be aware of the fact that the choice of other modules could increase coordination and teams' efforts to design the system. The product architectures can deeply influence the design process structure since the product modules should be designed independently by different design teams [11].
In this paper, we intentionally apply the method on two product domains only. More generally, this method may be used similarly by starting from another DMM, that is, product – tasks, tasks – team DMM. The axioms proposed in this work assume that if one SF is coupled to a group of components then these components are all inter-related, this assumption is not always true since one direct coupling between two components can be replaced by indirect coupling through one or more components.
Finally, different further works are envisaged. Currently, the DSM optimization of the two domains remains separate. We intend to adapt the clustering algorithm to jointly optimize the two DSM and to enhance the modularization. We aim at providing the system architect with an integrated tool that allows him to simulate and structure the different project domains in a concurrent manner. In the preliminary design stages ("planning and clarifying the task", "conceptual design" [22]), the system architect who plays the role of project manager too has to design concurrently the preliminary product architecture and the overall project structure, related to the embodiment and detail design stages.

References

[1] Ulrich K.T. The Role of Product Architecture in the Manufacturing Firm. Research Policy, 24, 1995, pp.419-440.

[2] Whitfield R.I., Smith J.S. and Duffy A.H.B. Identifying Component Modules. In Proceedings of Seventh International Conference on Artificial Intelligence in Design (AID'02), Cambridge, United Kingdom, 15-17 July 2002, pp. 571-592.

[3] Hölttä-Otto K. Modular product platform design. PhD, Helsinki University of Technology, Espoo, Finland, ISBN 951-22-7766-2, 2005.

[4] Jiao J., Simpson T., Siddique Z. Product family design and platform-based product development: a state-of-the-art review. Journal of intelligent manufacturing, special issue on Product family design and platform-based product development, 18(1), pp. 5-29, 2007..
[5] Browning T.R. Applying the design structure matrix to system decomposition and integration problems. IEEE Trans. Eng. Mgt., 2001, 48, pp. 292-306.
[6] Sharman D., Yassine A. Characterizing Complex Product Architectures. Systems Engineering Journal, 2004, 7(1), pp. 35 60.
[7] Ulrich K.T. and Eppinger S.D. Product Design & Development, 2000 (2nd Ed. McGraw-Hill, New York).
[8] Stone R., Wood K. and Crawford R. A Heuristic Method for Identifying Modules for Product Architectures. Design Studies, 21, 2000, pp.5-31.
[9] Kusiak A. and Huang C.C. Development of Modular Products. IEEE Transactions on components, packaging and manufacturing technology – part A, 19(4), 1996, 523-538.
[10] Pimmler T. U. and Eppinger S.D. Integration Analysis of Product Decompositions. In Proc. ASME Design Theory and Method Conference (DTM'94), Vol 68, 1994, pp. 343-351
[11] Sosa M., Eppinger S. and Rowles C. Identifying modular and integrative systems and their impact on design team interactions. Transactions of the ASME, 125, June 2003, pp. 240-252
[12] Yu T., Yassine A. and Goldberg D. Genetic algorithm for developing modular product architectures. In Proc ASME 15th Int Conf Des Theory Methodol, Chicago, September 2003.
[13] Liu Y., Chakrabarti A. and Bligh T. P. Transforming Functional Solutions into Physical Solutions. Proceedings of the ASME Design Theory and Methodology Conference, DETC/DTM-8768, Las Vegas, NV, 1999.
[14] Gu P., and Sosale S. Product modularization for life cycle engineering. Robotics and Computer Integrated Manufacturing, 15, 1999, pp.387-401
[15] Clarkson P. J., Simons C., Eckert C. Predicting Change Propagation in Complex Design. Proceedings ASME DETC 2001, Paper No DETC2001/DTM-21698, Pittsburgh, PA, USA.
[16] Suh N. P. Principles of Design, 1990 (Oxford University Press, Cambridge, UK).
[17] Akao Y. Quality Function Deployment, QFD - Integrating Customer Requirements into Product Design, 1990 (Productivity Press, Portland, ON, USA).
[18] Strawbridge Z., McAdams D., and Stone R. A Computational approach to conceptual design. Proc. of DETC 02, Montreal, Canada, September 29-October 2, 2002.
[19] Danilovic M., Browning T.R. Managing complex product development projects with design structures matrices and domain mapping matrices. International Journal of Project Management, 25, p300-314, 2007.
[20] Eppinger S.D., and Salminen V. Patterns of product development interactions. Int. Conf. on Engineering Design, ICED 01, Vol. 1, Glasgow, August 21-23, 2001, pp.283-290
[21] Steward R. P. and Donald V. The Design Structure System: A Method for Managing the Design of Complex Systems. IEEE Transactions on Engineering Management, 1981, 28(3), 71-74.
[22] Pahl G., Beitz W. Engineering Design: a Systematic Approach, 1996 (2nd Ed., Springer-Verlag, London).
[23] Dubois D., Prade H. Fuzzy sets and systems: Theory and applications. Academic Press 1980.
[24] Hartigan J. Clustering Algorithms, 1975 (Wiley & Sons, New York, NY).
[25] Thebeau R. Knowledge management of system interfaces and interactions for product development processes. Master of Science Thesis, MIT, 2001.
[26] Idicula J. Planning for concurrent engineering. Thesis draft, Nanyang Technology University, 1995.